УДК 004.89

ОБРАБОТКА РЕКУРСИВНЫХ ДАННЫХ КОНЕЧНЫМИ АВТОМАТАМИ

В.Б. Новосельцев, В.В. Соколова

Институт «Кибернетический центр» ТПУ
E-mail: vbn@osu.cctpu.edu.ru

Предлагается альтернативный стандартному формализму подход к обработке рекурсивных данных. Основной идеей является первичность функциональности, а не данных. Функциональность реализуется механизмами, основанными на теории конечных автоматов. Полученные результаты позволяют использовать новые методы для создания естественного описания предметной области, содержащей рекурсивные структуры, что, в свою очередь, повышает эффективность манипулирования такими данными.

Классическая парадигма работы с организованными данными определяется реляционной моделью Кодда [1]. При всей своей строгости и лаконичности эта модель обладает одним серьезным недостатком – она является существенно эксплицитной, т. е. не допускает «динамики» в процессе использования. Существует много рекурсивных моделей, в которых образующим элементом является определенная внутренняя структура, сама являющаяся объектом модели, например, модели геномов, сложные химические образования или экономические модели. В таких случаях, реляционный подход позволяет определять рекурсию на данных путем введения, например, явного понятия предка, однако, этот паллиативный вариант не более чем моделирует частичный порядок внесистемными конструкциями [2]. Определенное продвижение в организации рекурсивных данных может предоставить функциональный подход на основе λ-формализма Черча [3], реализованного, в частности, в семействе функциональных языков программирования класса ЛИСП, естественно включающих рекурсию, как базовую конструкцию управления.

В рамках предлагаемого подхода основная роль отводится функциональной стороне. Считается, что любые данные появляются в результате интерпретации первичной составляющей модели, либо применения некоторого автомата к другим данным. Таким образом, необходимо переопределить понятие запроса. *Атомарный запрос* есть первичное значение, допустимое в заданной интерпретации. *Непервичный (автоматный) запрос* определяется некоторым конечным автоматом и реализуется применением этого автомата к входным данным.

Входной поток для фиксированного запроса (автомата) представляет собой список, сформированный другими атомарными либо автоматными запросами. Термины *запрос* и *автомат* считаются синонимами, если не оговорено противного.

Различаются явное описание запроса и его реализация. Согласно классическому определению, конечный автомат – это система с ограниченным набором состояний и определенной дисциплиной переходов [4], поэтому любая система, обладающая ограниченным набором возможностей, каждая из которых может быть отмечена отдельным состоянием, может быть представлена в виде конечного автомата (КА).

Для строгого введения последующих понятий зафиксируем сигнатуру:

$$\Sigma = \langle A, E, T \rangle. \quad (1)$$

Здесь A – множество символов атрибутов, E – множество символов состояний, T – множество символов доменов (возможных типов). Считается, что в T зафиксировано непустое подмножество $D \subset T$ символов первичных доменов ($D \neq \varnothing$), содержащих базовые значения атрибутов (например, integer, date, text и т. д.). Считается, что при любой интерпретации первичному домену сопоставляется перечислимое множество элементов.

С учетом заданной сигнатуры (1) непервичный автомат Q определяется следующим образом:

$$Q \rightleftharpoons (A, R, \Delta, \sigma, F), \quad (2)$$

где $Q \in T \backslash D$ и Q не является именем первичного домена; A – алфавит над декартовым произведением $A \times T$, то есть это все комбинации допустимых пар из декартова произведения типов и имен атрибутов; $R \in E$ – конечное множество состояний автомата; Δ – функция перехода, определенная на множестве $A \times R$; $\sigma \in R$ – начальное состояние; $F \subseteq R$ – непустое множество конечных состояний.

В каждый фиксированный момент времени (такт) конечный автомат находится только в одном из возможных состояний, при этом число состояний, в которых может находиться конечный автомат – конечно. Автомат последовательно считывает элементы из входной строки. Каждый считанный символ либо переводит автомат в новое состояние, либо оставляет его в прежнем, одновременно КА генерирует элемент в выходную строку.

Функция перехода, ассоциированная с автоматом Q (2), является частичной функцией $\Delta: A \times R \rightarrow A \times R$. Областью определения данной функции является декартово произведение атрибутов и состояний, допустимых для конечного автомата Q, а множеством значений – преобразованные конечным автоматом пары <*элемент_алфавита, состояние*>. При необходимости подчеркнуть связь Δ с автоматом Q используется запись Δ^Q.

Считается, что функция перехода, как правило, дискретна, т. е. может быть задана множеством пар:

$$\Delta \rightleftharpoons \{(\alpha, \delta), (\alpha', \delta')\}. \quad (3)$$

На рис. 1 данным парам (3) соответствуют x – текущий анализируемый атрибут, y – атрибут, который автомат записал в выходной поток после обработки, и переход между состояниями $\delta \rightarrow \delta'$.

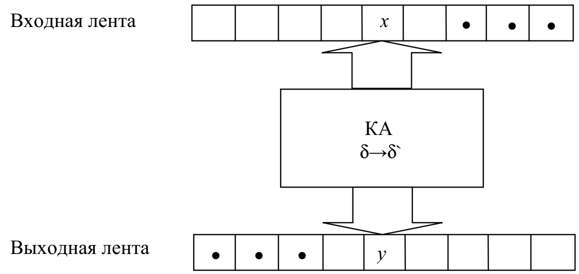


Рис. 1. *Структурная схема конечного автомата*

Входом и выходом любого автомата являются регулярные списки, которые могут быть вложенными. Двухуровневые списки, при соблюдении некоторых условий, сопоставляются плоским таблицам реляционного подхода – ниже рассматриваются именно такие списки. Списки большей вложенности также не запрещены, так, трехуровневые структуры могут использоваться при моделировании темпоральных или интенсиональных баз данных.

Введем понятие атрибута, которое понадобится для более строгого определения входного и выходного потоков автомата и других необходимых структур. Будем различать *имя* атрибута и его *значение (интерпретацию)*, а также *первичные (примитивные)* и *непервичные* атрибуты. Прежде всего, определим рекурсивно понятие *допустимого имени*.

Пусть $a \in A$, α – допустимое имя, тогда a и $\alpha.a$ – также допустимые имена, во втором случае α и a называются префиксом и суффиксом, соответственно. Допустимое имя α становится *именем атрибута*, только если оно связывается с некоторым доменом $T \in$ T. Факт связывания отмечается записью $T(\alpha)$ или $\alpha.T$ (точечная пара, согласно нотации языка ЛИСП) – первая форма подчеркивает функциональность связывания, а вторая – его ссылочность. Если $T \in D$, и α – примитивный атрибут, то $\alpha.T$ заменяется на α^T либо на α, когда ссылка на домен не существенна или очевидна из контекста. Связь именованного объекта с интерпретацией I изображается стандартной записью $\langle \textit{имя} \rangle|_I$. Интерпретация задается традиционно: примитивному атрибуту сопоставляется константа из первичного домена $\alpha^T|_I \in T|_I$, тогда автоматной функции Δ^Q соответствует отображение $\Delta^Q|_I: A|_I \times R|_I \rightarrow A|_I \times R|_I$.

Любой непервичный домен T, где $T \in \mathrm{T} \backslash D$ является структурой над *именованными* компонентами:

$$a.T \rightleftharpoons a.(a_1.T_1, ..., a_n.T_n), \quad (4)$$

где a – параметр-модификатор имен элементов списка согласно точечной нотации, a, $a_i \in$ A – попарно-различные, $T \in \mathrm{T} \backslash D$, $T_i \in$ T. В определении домена допускается рекурсия (в том числе, и неявная).

Интерпретация $\alpha.T|_I$ определяет пустой или сколь угодно длинный (но конечный) *регулярный* список вида $\alpha.((a_1.t_{11}|_I, ..., a_n.t_{n1}|_I)...(a_1.t_{1k}|_I, ..., a_n.t_{nk}|_I))$, где для всех значений индексов $i=1,...,n$, $j=1,...,k$, $a_i \in$ A, $t_{ij}|_I \in T_i|_I$, $T_i \in$ T. Регулярность списка определяется тем, что различные t_{ij} и t_{im} принадлежат одному и тому же домену T_i (4). Префикс может быть продолжен в скобочные фрагменты на любую глубину, так что $\alpha.((a_1.t_{11}...)...) =_{def} (\alpha.(a_1.t_{11}...)...) =_{def} ((\alpha.a_1.t_{11}...)...)$. Такие структуры можно считать *таблицами*, подразумевая, что подсписки соответствуют строкам (кортежам) таблиц реляционного подхода. Ясно, что t_{ij}, в свою очередь, могут являться списками, возможно, пустыми.

Рассмотрим реализационные аспекты, использующие функциональный аппарат системы программирования ЛИСП [5]. Зададим описание автомата λ-термом в стиле функциональных языков вида:

$$Q = (\lambda x.\Delta[x]). \quad (5)$$

Как было выше указано (2), $Q\in T\backslash D$, где Q – имя автомата не может быть именем из первичного домена, а $[a]$, определяемое формулой (3), традиционно обозначает вхождение (элементов) списка в форму Δ.

В наиболее общем случае структура формы Δ определяется следующей конструкцией согласно нотации языка ЛИСП:

(*DO* (<*init_form*>) (<*stop_condition*> <*actions*>) <*actions*>). (6)

Интерпретацией автомата Q (5), как отмечалось выше, является применение его к списку *S интерпретированных* атрибутов – это обозначается традиционной для функционального подхода записью $(QS|_{I})$ или (QS), если интерпретация очевидна из контекста). Результатом интерпретации автомата является также список (возможно, пустой), который, в свою очередь, может использоваться в качестве домена при спецификации нового атрибута. Поскольку атрибуты S могут являться применениями автоматов, возникает вопрос о трактовке операции суперпозиции автоматов, который решается через вложенный функциональный вызов.

Рассмотрим ряд дополнительных соглашений относительно рекурсивных структур данных. Пусть регулярный список L некоторым образом порождается доменом $T_0=(a_1.T_1,...,a_n.T_n)$, состоящим из первичных атрибутов. Условимся рассматривать только такие списки, у которых первый атрибут связан с первичным доменом, то есть $T_1\in D$ (*синтаксическое ограничение*). Соответственно, первый атрибут $(a_1.T_1)$ будем называть *первичным ключом* и считать его *уникальным именем* любого подсписка списка L верхнего уровня, так что в $L=\alpha.((a_1.t_{11}|_{I},...,a_n.t_{n1}|_{I})...(a_1.t_{1k}|_{I},...,a_n.t_{nk}|_{I}))$ значения всех $a_1.t_{1i}|_{I}$ – попарно-различны, т. е. уникальны (семантическое ограничение).

Множество $K_L \rightleftharpoons \{a_1.t_{1i}\ |_{I}\}$ (7)

будем называть *ключевым множеством* списка L. Считается, что на элементах K определено отношение полного порядка ("$<k$"), как правило, индуцируемое соответствующим отношением на T_1, а также операция инкрементирования. Важно подчеркнуть, что в набор параметров автомата x (5) определения автомата в качестве обязательного параметра входит системный ключ a_1. Отметим, что отношения порядка могут быть определены на домене любого атрибута (группы атрибутов), – это позволяет организовывать необходимые индексированные цепочки, что аналогично понятию ключа в реляционном подходе.

Подчеркнем еще раз, что любой домен является порождением автомата, при этом первичные домены трактуются, как реализации *is-a* отношений (автоматов) вида (*is-a integer x*), доставляющих истину, если предъявленный объект распознается корректно.

Переопределим понятие таблицы данных в предложенных соглашениях. Для этого зададим понятие схемы таблицы для объекта данных r – это поименованный список тэгов вида: $T(r)=r.(...,T_i(a_i),...)$. При рассмотрении набора таблиц допускается появление неявной или опосредованной рекурсии, причем требование присутствия в описании каждой схемы хотя бы одного первичного атрибута гарантирует конечность реализации любой таблицы. Для «свертки данных» используется автомат-конструктор: $(CONSTR(T(a_1,...,a_n)))=_{\mathrm{def}}(T(a_1,...,a_n))$. Во внешней переменной запоминается прохождение всех рекурсивных ответвлений при обработке данных, что сокращает время обработки.

При реализации существенным образом используется теория функциональных моделей, в частности, из нее заимствованы обозначения [6]. Определим обратную операцию – *развертку данных на непервичном атрибуте*. Пусть $T(r)=r.(...,T_{ij}(a_{ij}),...)$ – описание домена (4) и домен $T_{ij}\notin D$. Развёрткой домена T на атрибуте a_{ij} будем называть список, получающийся в результате подстановки в описание исходного домена на место имени $T_{ij}(a_{ij})$, раскрытого соответствующим автоматом, подсписка-домена $T_{ij}(a_{ij})$. Все добавленные в развернутый домен T в результате такого действия атрибуты модифицируются префиксом $r.a_{ij}$, что индуцирует и модификацию автомата, доставляющего реализацию домена $T_{ij}(a_{ij})$. Процесс построения развёртки на атрибуте может повторяться многократно – пока имеются атрибуты, связанные с неэлементарными схемами.

Зададим операцию полной развертки**.** Под *полной развёрткой* домена T понимается завершенный процесс применения соответствующего автомата к интерпретации домена. Полная развёртка реализует систему строго вложенных стеков, которая для рекурсивных доменов, вообще говоря, может оказаться неограниченной. Для сохранения конструктивности построений как раз и вводятся понятия *первичного ключа* (7) и введенное понятие *замыкания атрибутов* относительно автоматного домена (запроса).

Пусть T – автоматный (возможно, рекурсивный) домен и пусть A – некоторый набор атрибутов домена. Замыканием A^+ для A относительно T будем называть объединение всех таких атрибутов X, что автоматное отображение $Q:A\rightarrow X$ может быть получено из информации о домене T. Здесь Q – (автоматный) терм в сигнатуре Σ, построенный по специальным правилам, A и X – непустые наборы атрибутов. Замыкание A^+ используется для обеспечения семантической необходимости выхода из рекурсии в практических случаях.

В предлагаемом подходе (6) результат обработки произвольного списка вычисляется «по частям», т. е. вместо вычисления полного списка $\alpha.(a_1.t_1,a_2.t_2,...,a_n.t_n)$ можно найти первый элемент $\alpha.(a_1.t_1)$ и выполнить вычисления остальных элементов с помощью автоматов, реализующих конкретные операции. При этом получается принципиальная экономия памяти ценой незначительно-

го перерасхода времени на вспомогательное построение. *Автоматная операция* – это просто ссылка на уже существующую (ранее определенную) программу, связанную с контекстом ее исполнения (вставка, удаление или выборка), т. е. с состоянием ассоциативного списка в момент его построения.

Для повышения эффективности работы автоматов, введем специальные операции ‖ – *приостановка* вычислений и @ – *возобновление* ранее отложенных вычислений. Техника приостановки и возобновления функций может быть названа *вызовом по необходимости*.

Например, для построения ряда целых от *M* до *N* с последующим их суммированием, выполним следующее:

(defun ряд_цел (M N) (cond ((> M N) nil)

(T(cons M (|| (ряд_цел (1+ M) N))))))

(defun sum (X) (cond ((= X 0) 0)

(T (+ (car X)(@ (sum (cdr X))))))),

где *car* – первый элемент из активного значения списка, *cdr* – активное значение списка без первого элемента, *cons* – формирование узла по двум верхним значениям списков, *cond* – условие завершения вычислений, *nil* – терминальный элемент списка, *sum* – автомат, выполняющий операцию суммирования чисел.

С целью исключения повторного вычисления совпадающих списков, в его внутреннее представление вводится логическая переменная (flag), имеющая значение True (истина), для уже выполненных шагов прохождения рекурсии, и False (ложь) – для невыполненных.

Данный автомат-операция позволяет обрабатывать неограниченные рекурсивно-вычислимые множества. Например, можно работать с рядом целых чисел, больших чем *N*:

(defun цел (M) (cons M (|| (цел (1+ M))))).

Из организованного таким образом списка, можно выбирать нужное количество элементов; например первые *K* элементов можно получить применением функции:

(defun первые (K Int)

(cond ((= Int Nil) Nil)((= K 0) Nil)

(T (cons (car Int)(первые (@

(cdr Int))))))).

Эффект таких приостанавливаемых и возобновляемых вычислений получается путем следующей реализации операций ‖ и @:

||e = > (lambda () e),

@e = > (e),

что при интерпретации приводит к связыванию функционального аргумента с ассоциативным списком для операции ‖ и к вызову функции выполнения *EVAL* для операции @.

Таким образом, предложенная методология может использоваться в качестве теоретического аппарата для построения систем управления базами данных, поддерживающих естественное представление и механизмы манипулирования рекурсивными структурами, а применение автоматного подхода позволяет повысить эффективность обработки таких данных, поскольку ни одна существующая система менеджмента данных не имеет внутреннего аппарата для работы с рекурсивными структурами.

Работа выполнена на основе индивидуального гранта ТПУ молодым ученым на проведение научных исследований (2006 г.).

СПИСОК ЛИТЕРАТУРЫ

1. Ульман Дж. Основы систем баз данных. – М.: Финансы и статистика, 1983. – 334 с.
2. Новосельцев В.Б., Соколова В.В. Расширение алгебры Кодда операциями с рекурсивными объектами // Вестник ТГУ. Серия «Математика. Кибернетика. Информатика». – 2004. – № 284. – С. 18–21.
3. Черч А. Введение в математическую логику. – М.: Изд-во иностранной литературы, 1960. – 488 с.
4. Карпов Ю.Г. Теория алгоритмов и автоматов. – СПб.: Геликон Плюс, 2000. – 256 с.
5. Хювенен Э., Сеппянен Й. Мир Лиспа. В 2-х т. Т. 1: Введение в язык Лисп и функциональное программирование. – М.: Мир, 1990. – 447 с.
6. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47. – № 5. – С. 1014–1030.